Class is 170.

Announcements

Matlab Grader homework,
1 and 2 (of less than 9) homeworks Due 22 April **tonight,** Binary graded.
167, 165,164 has done the homework. (**If you have not done HW talk to me/TA!**)
Homework 3 due **5 May**
Homework 4 (SVM +DL) due ~24 May

Jupiter "GPU" home work released Wednesday. Due 10 May

Projects: 41 Groups formed. Look at Piazza for help.
Guidelines is on Piazza
May 5 proposal due. TAs and Peter can approve.
Email or use dropbox
https://www.dropbox.com/request/XGqCV0qXm9LBYz7J1msS
Format "Proposal"+groupNumber
May 20 presentation

Today:

- Stanford CNN 11, SVM, (Bishop 7)
- Play with Tensorflow playground before class http://playground.tensorflow.org
  Solve the spiral problem

Monday

- Stanford CNN 12, K-means, EM (Bishop 9),

Mike Bianco

Projects

- **3-4** person groups preferred
- Deliverables: Poster, Report & main code (plus proposal, midterm slide)
- **Topics** your own or chose form suggested topics. Some **physics inspired**.
- **April 26 groups** due to TA.
- **41 Groups formed. Look at Piazza for help.**
- **Guidelines is on Piazza**
- **May 5** proposal due. TAs and Peter can approve.

Email or use dropbox Format “Proposal”+groupNumber

https://www.dropbox.com/request/XGqCV0qXm9LBYz7J1msS

- **May 20** Midterm slide presentation. Presented to a subgroup of class.
- **June 5** final poster. Upload June ~3
- Report and code due **Saturday 15 June.**

Confusion matrix/Wikipedia

If a classification system has been trained to distinguish between cats, dogs and rabbits, a confusion matrix will summarize the test results. Assuming a sample of 27 animals — 8 cats, 6 dogs, and 13 rabbits, the confusion matrix could look like the table below:

		Actual class		
		Cat	**Dog**	**Rabbit**
Predicted class	**Cat**	**5**	2	0
	Dog	3	**3**	2
	Rabbit	0	1	**11**

7
8
12
8 6 13 27

Cat:

		Actual class	
		Cat	**Non-cat**
Predicted class	**Cat**	5 True Positives	2 False Positives
	Non-cat	3 False Negatives	17 True Negatives

27-5-3-2

11 + 3 + 2 + 1

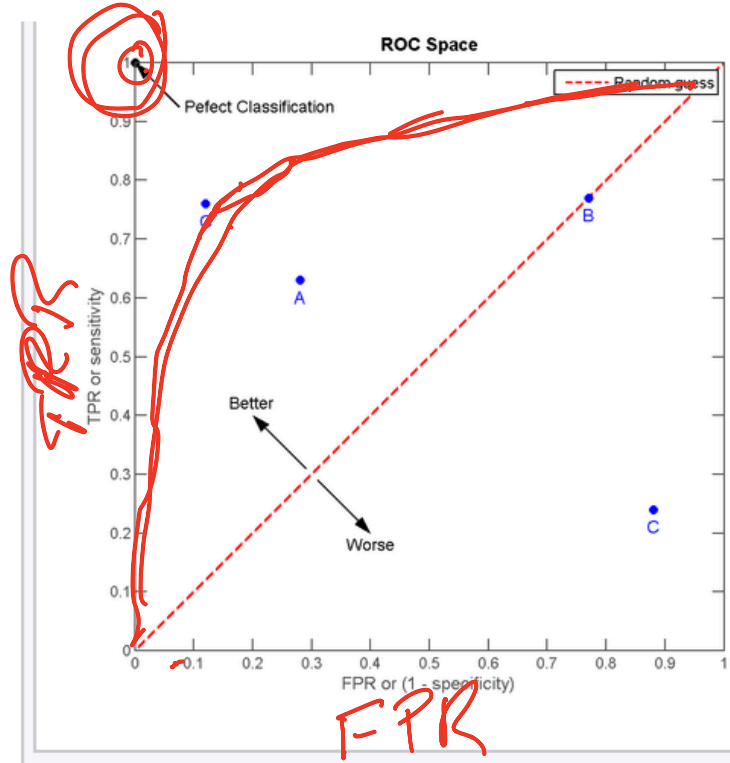
Let us define an experiment from **P** positive instances and **N** negative instances for some condition. The four outcomes can be formulated in a 2×2 *confusion matrix*, as follows:

		True condition				
	Total population	Condition positive 8	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**, Power 5	**False positive**, Type I error 2	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error 3	**True negative** 17	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR–) $= \frac{\text{FNR}}{\text{TNR}}$		

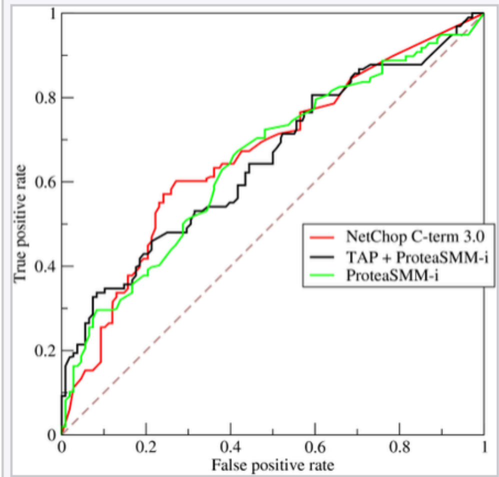
$TPR = \frac{5}{8}$ $FPR = \frac{2}{19}$

Recall

ROC curve (receiver operating charateristic)



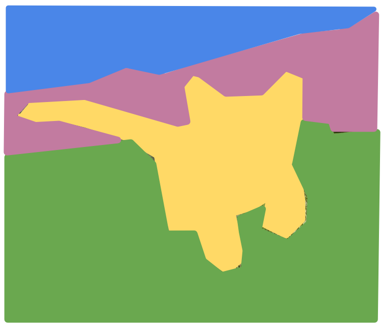
The ROC space and plots of the four prediction examples.



ROC curve of three predictors of peptide cleaving in the proteasome.

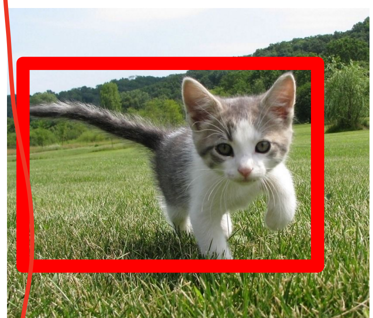
Other Computer Vision Tasks

Semantic Segmentation



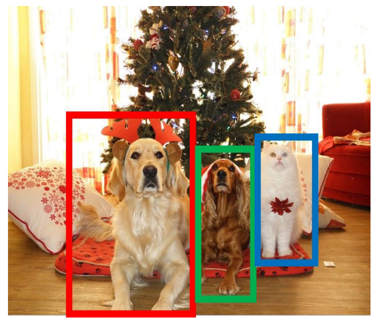
GRASS, CAT, TREE, SKY

Classification + Localization



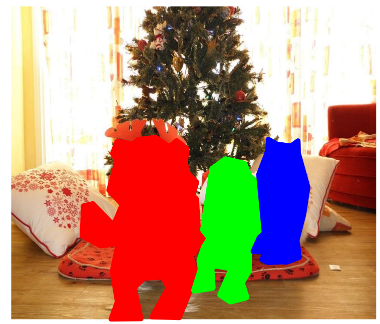
CAT

Object Detection



DOG, DOG, CAT

Instance Segmentation



DOG, DOG, CAT

No objects, just pixels

Single Object

Multiple Object

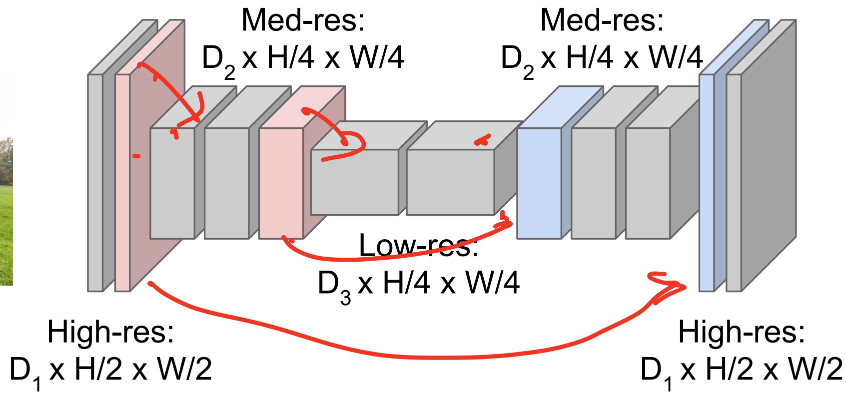
Semantic Segmentation Idea: Fully Convolutional

Downsampling:
Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling:
Unpooling or strided transpose convolution

Input:
$3 \times H \times W$



Predictions:
$H \times W$

In-Network upsampling: “Unpooling”

Nearest Neighbor

1	2
3	4

Input: 2 x 2

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Output: 4 x 4

“Bed of Nails”

1	2
3	4

Input: 2 x 2

1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Output: 4 x 4

In-Network upsampling: “Max Unpooling”

Max Pooling
Remember which element was max!

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

Input: 4 x 4

5	6
7	8

Output: 2 x 2

… Rest of the network

Max Unpooling
Use positions from pooling layer

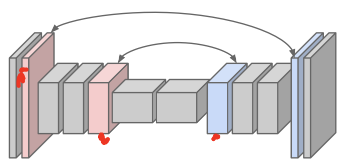
1	2
3	4

Input: 2 x 2

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

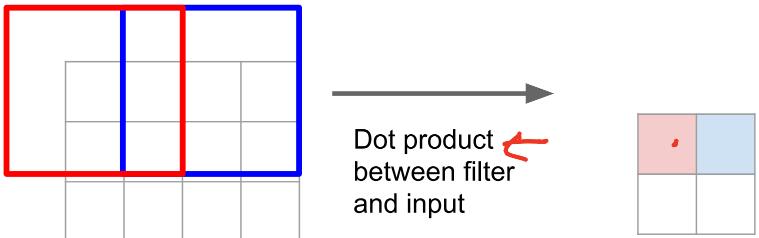
Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers



Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1

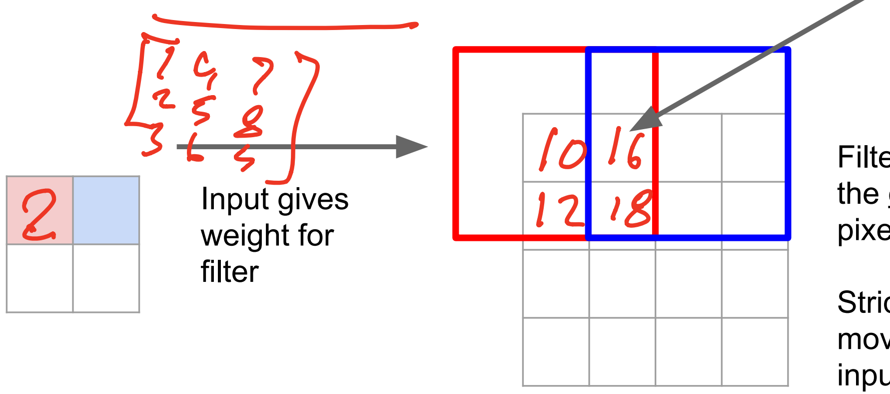


Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

Other names:
- -Deconvolution (bad)
- -Upconvolution
- -Fractionally strided convolution
- -Backward strided convolution

3 x 3 **transpose** convolution, stride 2 pad 1

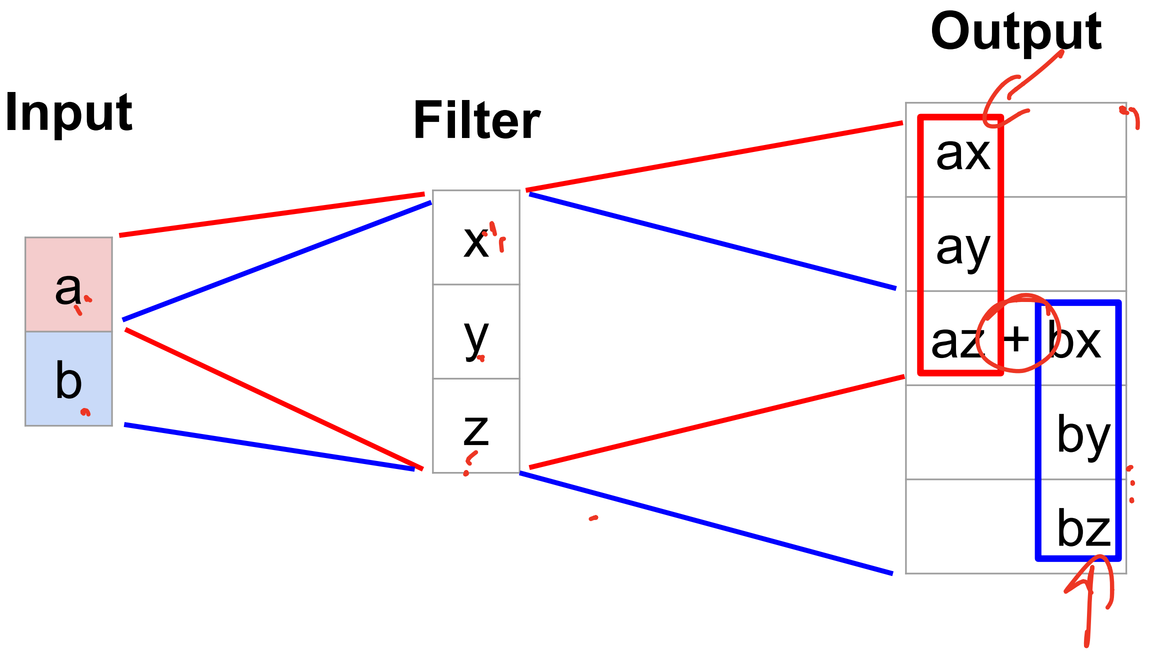


Sum where output overlaps

Filter moves 2 pixels in the output for every one pixel in the input

Stride gives ratio between movement in output and input

Transpose Convolution: 1D Example



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Need to crop one pixel from output to make output exactly 2x input

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

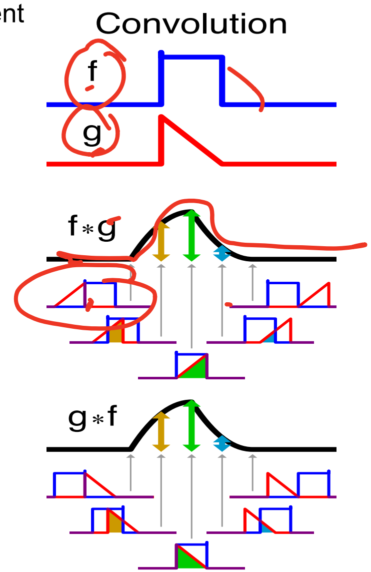
Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

When stride=1, convolution transpose is just a regular convolution (with different padding rules)



Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, <u>stride=2</u>, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

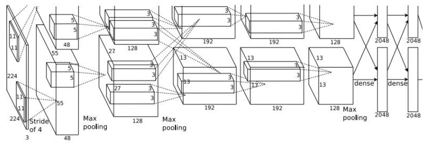
$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

When stride>1, convolution transpose is no longer a normal convolution!

Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

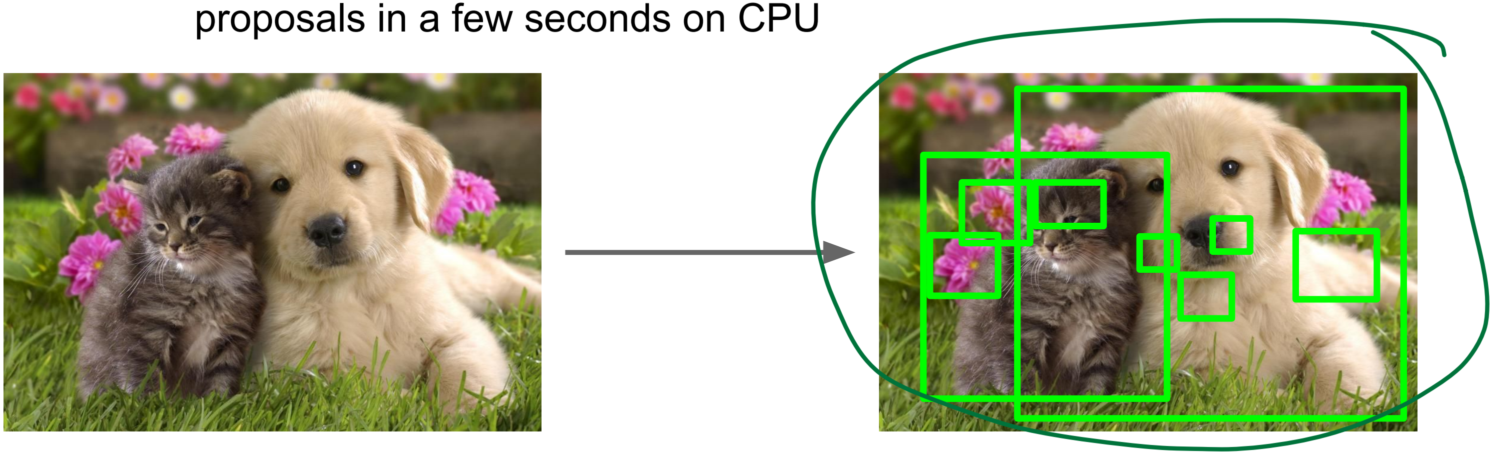


Dog? NO
Cat? YES
Background? NO

Problem: Need to apply CNN to huge number of locations and scales, very computationally expensive!

Region Proposals

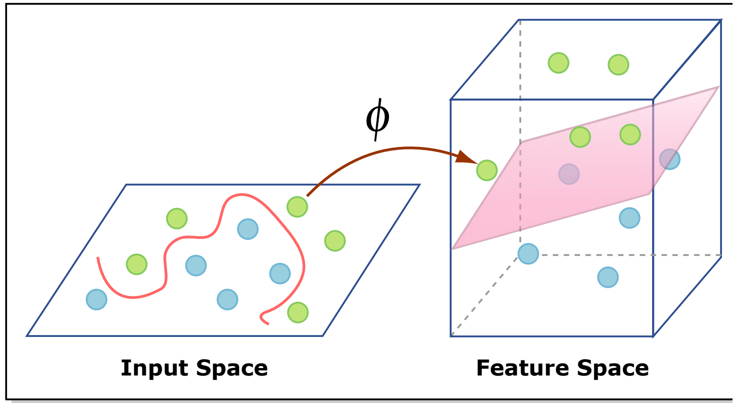
- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 1000 region proposals in a few seconds on CPU



Kernels

We might want to consider something more complicated than a linear model:

Example 1: $[x^{(1)}, x^{(2)}] \rightarrow \mathbf{\Phi}\left([x^{(1)}, x^{(2)}]\right) = \left[x^{(1)2}, x^{(2)2}, x^{(1)}x^{(2)}\right]$



Information unchanged, but now we have a **linear** classifier on the transformed points.

With the kernel trick, we just need kernel

$$k(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{\Phi}(\boldsymbol{a})^T\ \boldsymbol{\Phi}(\boldsymbol{b})$$

Dual representation, Sec 6.2

Primal problem: $\min_{\boldsymbol{w}} E(\boldsymbol{w})$ $\quad w \in R^M$

$$E = \frac{1}{2}\sum_n^N \{\boldsymbol{w}^T \boldsymbol{x}_n - t_n\}^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

Solution $\quad \boldsymbol{w} = \boldsymbol{X}^+\boldsymbol{t} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I}_M)^{-1}\boldsymbol{X}^T\boldsymbol{t}$

$$= \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T + \lambda \boldsymbol{I}_N)^{-1}\boldsymbol{t} = \boldsymbol{X}^T(\boldsymbol{K} + \lambda \boldsymbol{I}_N)^{-1}\boldsymbol{t} = \boldsymbol{X}^T\boldsymbol{a}$$

$a \in R^N$

The kernel is $\mathbf{K} = \boldsymbol{X}\boldsymbol{X}^T$

Dual representation is : $\min_{\boldsymbol{a}} E(\boldsymbol{a})$

$$E = \frac{1}{2}\sum_n^N \{\boldsymbol{w}^T \boldsymbol{x}_n - t_n\}^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \|\boldsymbol{K}\boldsymbol{a} - \boldsymbol{t}\|_2^2 + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a}$$

Prediction

$$y = \boldsymbol{w}^T\boldsymbol{x} = \boldsymbol{a}^T\boldsymbol{X}\boldsymbol{x} = \sum_n^N a_n \boldsymbol{x}_n^T\boldsymbol{x} = \sum_n^N a_n k(\boldsymbol{x}_n, \boldsymbol{x})$$

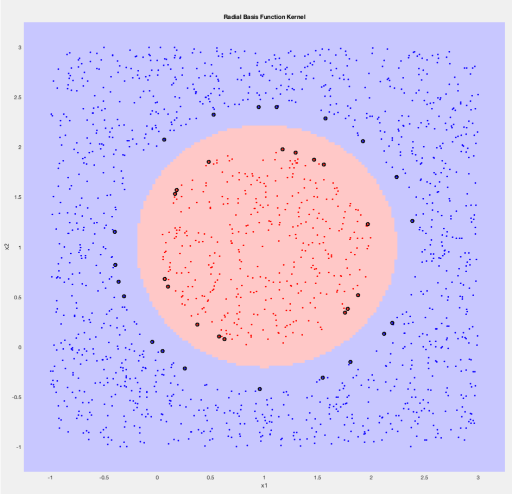
Dual representation, Sec 6.2

Prediction

$$y = \boldsymbol{w}^T \boldsymbol{x} = \boldsymbol{a}^T \boldsymbol{X} \boldsymbol{x} = \sum_n^N a_n \boldsymbol{x}_n^T \boldsymbol{x} = \sum_n^N a_n k(\boldsymbol{x}_n, \boldsymbol{x})$$

- Often **a is sparse (…** Support vector machines**)**
- We don't need to know **x or** $\boldsymbol{\varphi}(\boldsymbol{x})$. ***Just the Kernel***

$$E(\boldsymbol{a}) = \|\boldsymbol{K}\boldsymbol{a} - \boldsymbol{t}\|_2^2 + \frac{\lambda}{2} \boldsymbol{a}^T \boldsymbol{K} \boldsymbol{a}$$



Lecture 10
Support Vector Machines

M-NIST

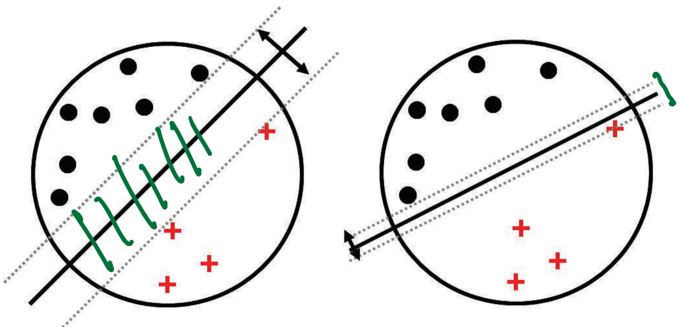
Non Bayesian!

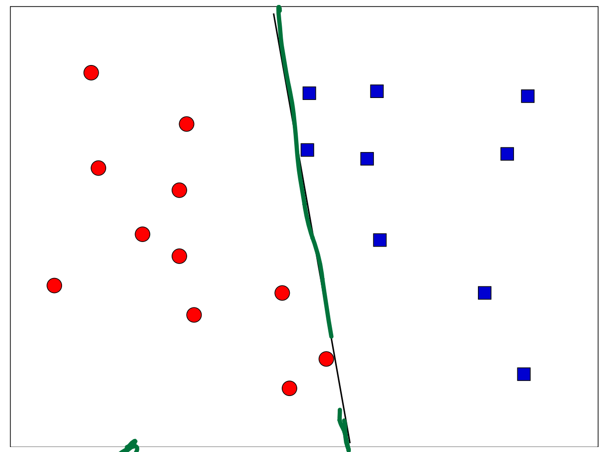
Features:

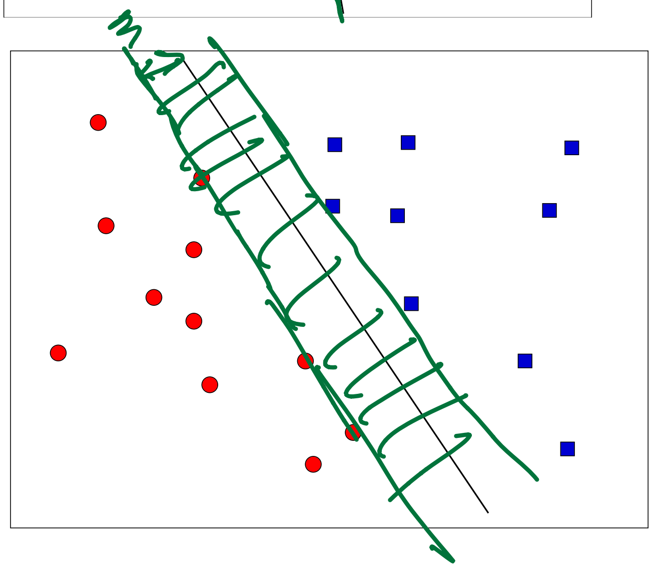
- Kernel
- Sparse representations
- Large margins

Regularize for plausibility

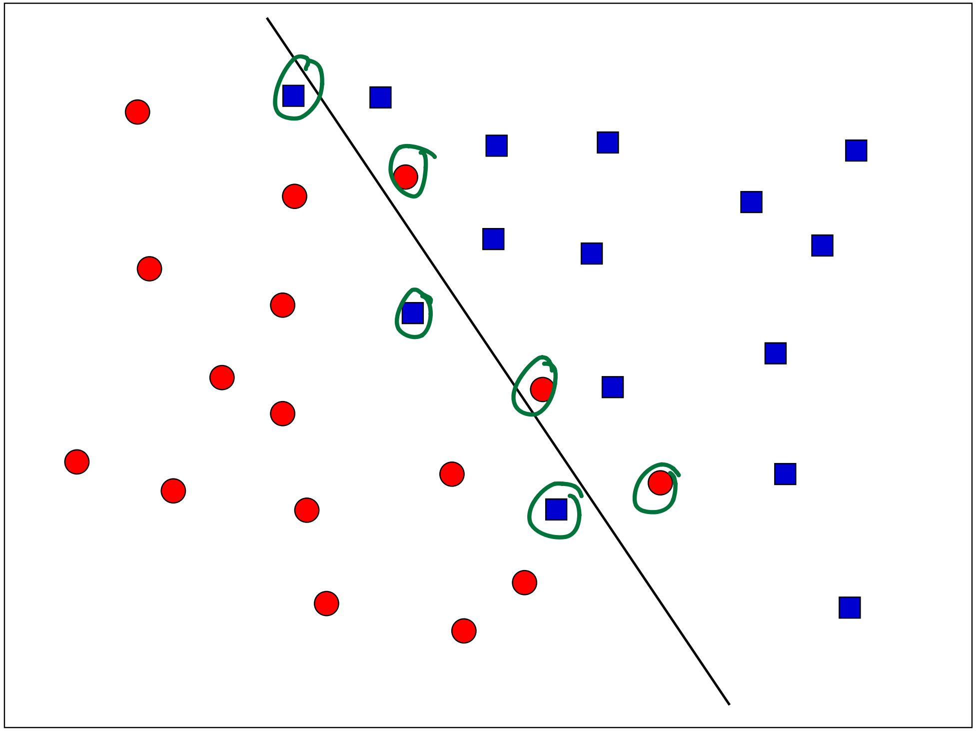
- Which one is best?
- We maximize the margin





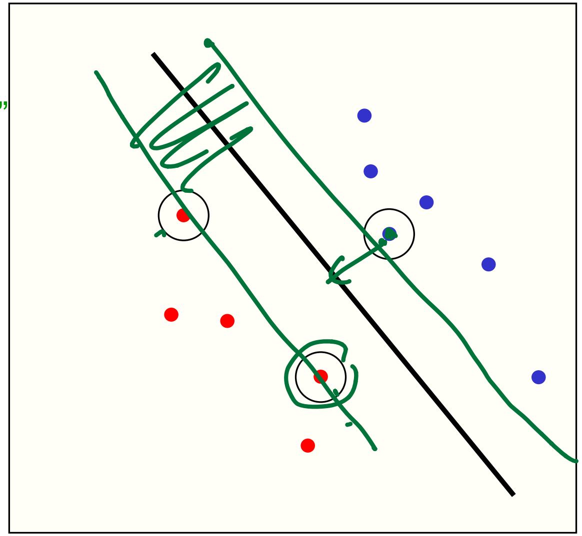


Regularize for plausibility



Support Vector Machines

- The line that maximizes the minimum margin is a good bet.
  - The model class of "hyper-planes with a margin *m*" has a low VC dimension if *m* is big.
- This maximum-margin separator is determined by a subset of the datapoints.
  - Datapoints in this subset are called "**support vectors**".
  - It is useful computationally if only few datapoints are support vectors, because the support vectors decide which side of the separator a test case is on.



The support vectors are indicated by the circles around them.

Lagrange multiplier (Bishop App E)

$\max(f(x))$ subject to $g(x) = 0$

Taylor expansion

$$g(\boldsymbol{x} + \boldsymbol{\varepsilon}) = g(\boldsymbol{x}) + \boldsymbol{\epsilon}^T \nabla g(\boldsymbol{x})$$

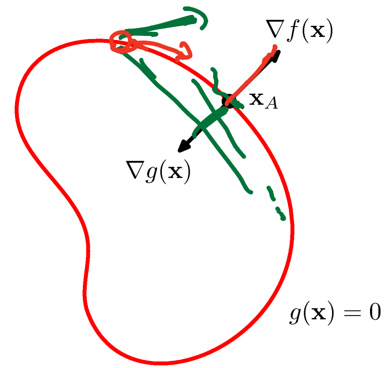
$$= 0 + \varepsilon^T \nabla g(x) + \dots$$

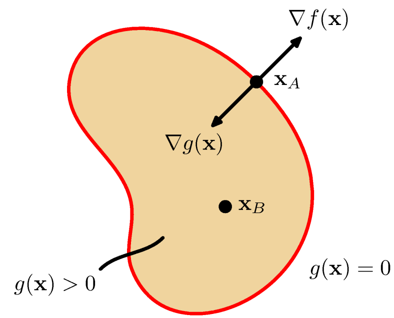
$$\nabla f + \lambda \nabla g$$

$$L(x, \lambda) = f(x) + \lambda g(x)$$

$$\nabla_x L = 0$$

$$\frac{\partial L}{\partial \lambda} = 0 = g(x)$$

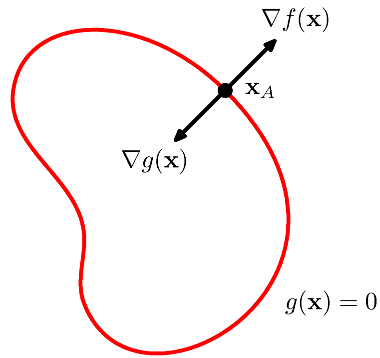




Lagrange multiplier (Bishop App E)

$$\max(f(\boldsymbol{x})) \text{ subject to } g(\boldsymbol{x}) > 0$$

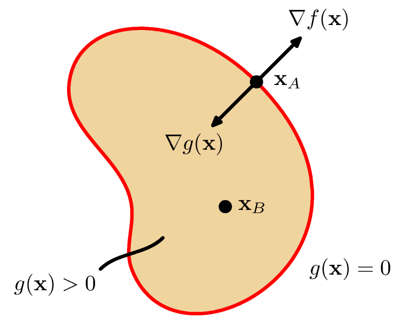
$$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda g(\boldsymbol{x})$$



Either $\nabla \mathrm{f}(\boldsymbol{x}) = \mathbf{0}$

Then $g(\boldsymbol{x})$ is inactive, $\lambda=0$

Or $g(\boldsymbol{x}) = 0$ *but* $\lambda > 0$



Thus optimizing $L(\boldsymbol{x}, \lambda)$ with the Karesh-Kuhn-Trucker (KKT) equations

$$g(\boldsymbol{x}) \geq 0$$

$$\lambda \geq 0$$

$$\lambda g(\boldsymbol{x}) = 0$$

Testing a linear SVM

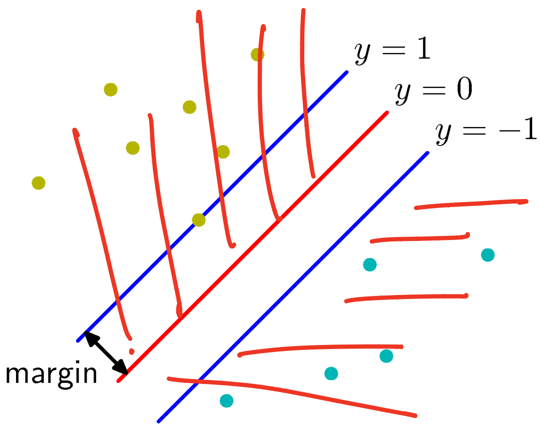
- The separator is defined as the set of points for which:

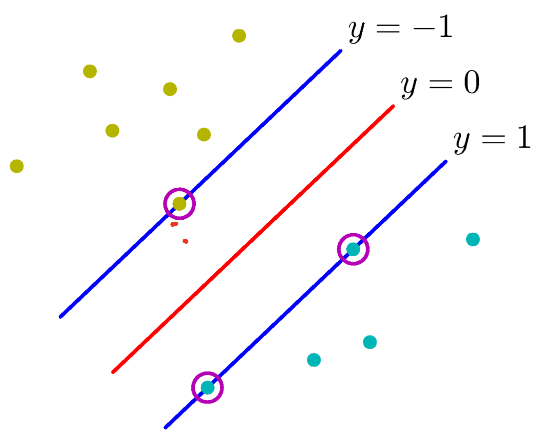
$w^T x + b$

$$y = \mathbf{w}.\mathbf{x} + b = 0$$

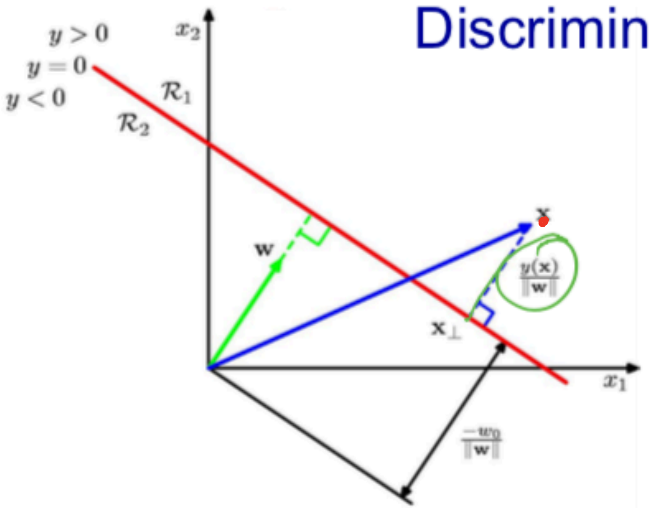
$$so\ if\ \ \mathbf{w}.\mathbf{x}^c + b > 0\ \ say\ its\ a\ positive\ case$$

$$and\ if\ \ \mathbf{w}.\mathbf{x}^c + b < 0\ \ say\ its\ a\ negative\ case$$





Discriminant functions



The planar decision surface in data-space for the simple linear discriminant function:

$$\mathbf{w}^T\mathbf{x} + w_0 \geq 0$$

$y = w^T x + w_0$

$y_\perp = w^T x_\perp + w_0 = 0$ ✓

$\|w\|_2^2 = w^T w$

X on plane => y=0 =>

Distance from plane

$x = x_\perp + r \frac{w}{\|w\|_2}$

$w^T x = w^T x_\perp + r \frac{w^T w}{\|w\|_2}$

$= -w_0 + r \|w\|_2$

$r = \frac{y}{\|w\|_2}$

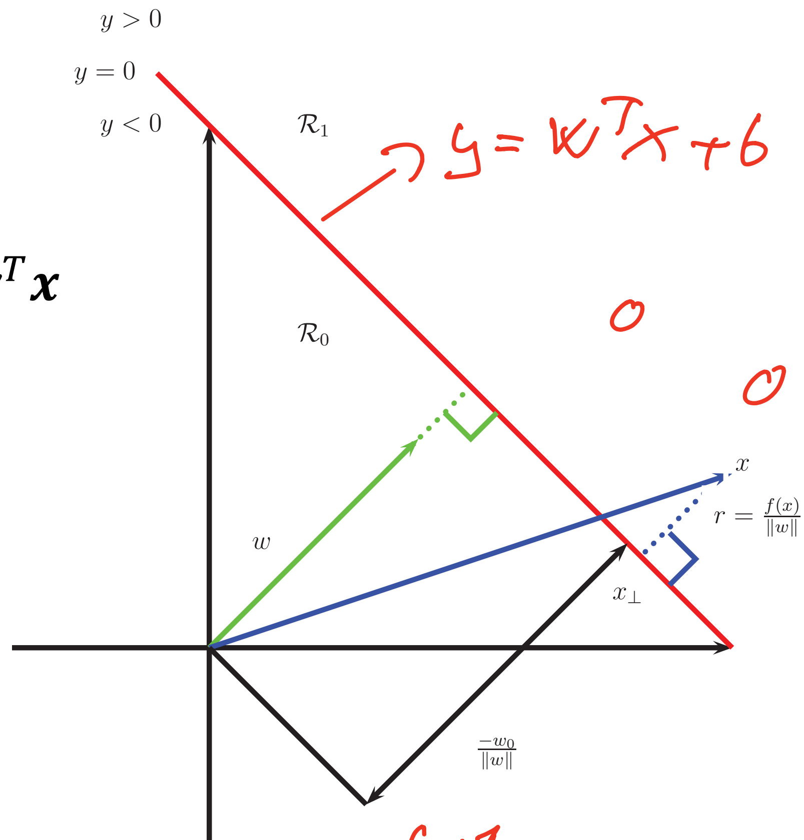
$x = x_\perp + \frac{y}{\|w\|_2}$

Large margin

$$y = \boldsymbol{w}^T\boldsymbol{x} + b$$

$$\boldsymbol{x}_n = \boldsymbol{x}_\perp + r_n \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$$

x on plane => y=0 => $b = -\boldsymbol{w}^T\boldsymbol{x}$



$$r_n = \frac{\boldsymbol{w}^T\boldsymbol{x}_n + b}{\|\boldsymbol{w}\|} = \frac{y_n}{\|\boldsymbol{w}\|}$$

$$t_n y_n \geq 1$$

$$\max_{\boldsymbol{w}} \frac{1}{\|\boldsymbol{w}\|} \min_n t_n y_n$$

$t_n = \begin{cases} +1 \\ -1 \end{cases}$

$x \to \phi(x)$

Maximum margin (Bishop 7.1)

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

Subject to

$$t_n\left(\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)+b\right) \geqslant 1, \qquad n=1,\ldots,N. \tag{7.5}$$

Lagrange function

$$L(\mathbf{w},b,\mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n\left\{t_n(\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)+b)-1\right\} \tag{7.7}$$

Differentiation

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) \tag{7.8}$$

$$0 = \sum_{n=1}^{N} a_n t_n. \tag{7.9}$$

Dual representation

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n,\mathbf{x}_m) \tag{7.10}$$

with respect to $\mathbf{a}$ subject to the constraints

$$a_n \geqslant 0, \qquad n=1,\ldots,N, \tag{7.11}$$

$$\sum_{n=1}^{N} a_n t_n = 0. \tag{7.12}$$

This can be solved with quadratic programming

Maximum margin (Bishop 7.1)

- KKT conditions

$$a_n \geqslant 0 \tag{7.14}$$

$$t_n y(\mathbf{x}_n) - 1 \geqslant 0 \tag{7.15}$$

$$a_n \{t_n y(\mathbf{x}_n) - 1\} = 0. \tag{7.16}$$

either $a_n = 0$ or $t_n y(\mathbf{x}_n) = 1.$

- Solving for a_n

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n) \tag{7.8}$$

- Prediction

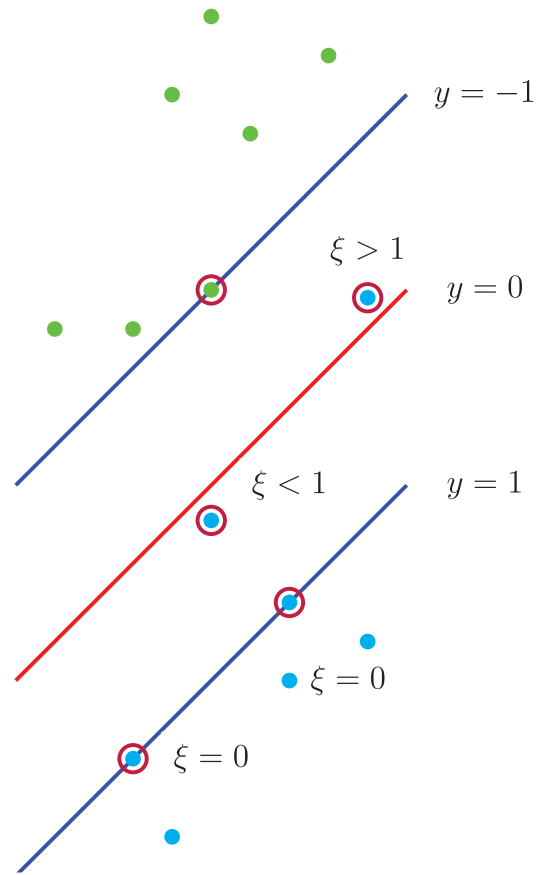
$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b. \tag{7.13}$$

$$k = \phi^T(x)\phi(x_n)$$

If there is no separating plane…

- Use a bigger set of features.
  - Makes the computation slow? "Kernel" trick makes the computation fast with many features.
- Extend definition of maximum margin to allow non-separating planes.
  - Use "slack" variables $\xi = |t_n - y(\boldsymbol{x}_n)|$

$$t_n y(\mathbf{x}_n) \geqslant 1 - \xi_n, \qquad n = 1, \ldots, N \tag{7.20}$$



Objective function

$$C \sum_{n=1}^{N} \xi_n + \frac{1}{2}\|\mathbf{w}\|^2 \tag{7.21}$$

SVM classification summarized--- Only kernels

- Minimize with respect to $\boldsymbol{w}, \mathrm{w}_0$

$$C \sum_n^N \zeta_n + \frac{1}{2} ||\boldsymbol{w}||^2 \qquad \text{(Bishop 7.21)}$$

- Solution found in dual domain with Lagrange multipliers
  - $a_n, n = 1 \cdots N$ and
- This gives the support vectors S

$$\hat{\boldsymbol{w}} = \sum_{n \in S} a_n t_n \boldsymbol{\varphi}(xn) \qquad \text{(Bishop 7.8)}$$

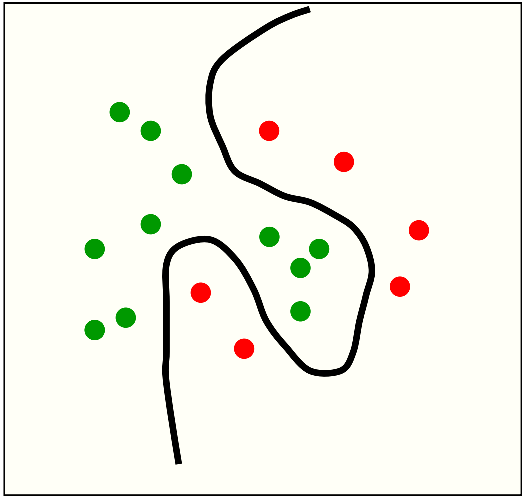
- Used for predictions

$$\hat{y} = \mathrm{w}_0 + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\varphi}(x) = \mathrm{w}_0 + \sum_{n \in S} a_n t_n \boldsymbol{\varphi}(x_n)^{\mathrm{T}} \boldsymbol{\varphi}(x)$$

$$= \mathrm{w}_0 + \sum_{n \in S} a_n t_n k(x_n, x) \qquad \text{(Bishop 7.13)}$$

How to make a plane curved

- Fitting hyperplanes as separators is mathematically easy.
  - The mathematics is **linear.**
- Replacing the raw input variables with a much larger set of features we get a nice property:
  - A planar separator in high-D feature space is a curved separator in the low-D input space.



A planar separator in a 20-D feature space projected back to the original 2-D space

SVMs are Perceptrons!

- SVM's use each training case, x, to define a feature K(x, .) where K is user chosen.
  - So the user designs the features.
- SVM do "feature selection" by picking support vectors, and learn feature weighting from a big optimization problem.
- =>SVM is a clever way to train a standard perceptron.
  - What a perceptron cannot do, SVM cannot do.

- SVM DOES:
  - Margin maximization
  - Kernel trick
  - Sparse

SVM Code for classification (libsvm)

Part of ocean acoustic data set http://noiselab.ucsd.edu/ECE285/SIO209Final.zip

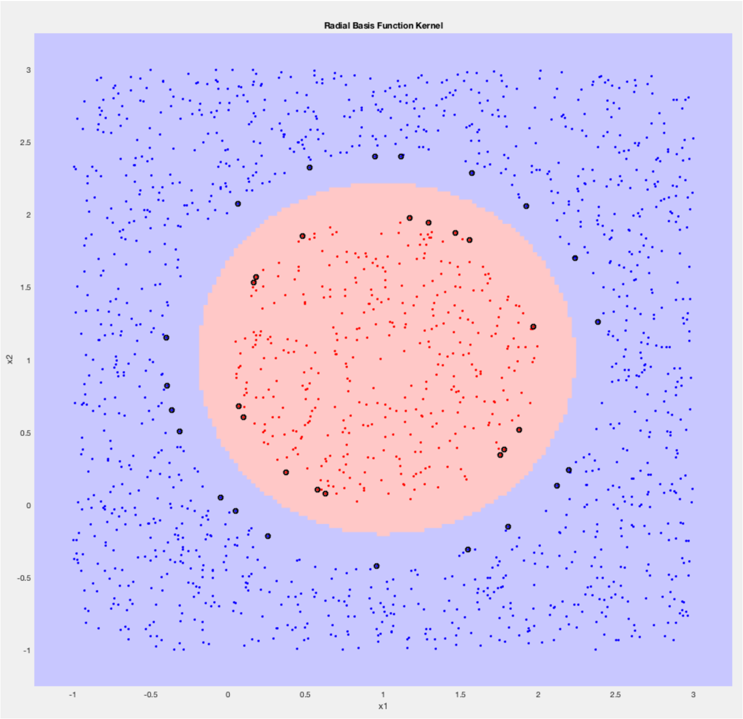
case 'Classify'

% train

model = **svmtrain**(Y, X,['-c 7.46 -g ' gamma ' -q ' kernel]);

% predict

[predict_label,~, ~] = **svmpredict(**rand([length(Y),1]), X, model,'-q');



>> modelmodel = struct with fields:

Parameters: [5×1 double]

nr_class: 2

totalSV: 36

rho: 8.3220

Label: [2×1 double]

sv_indices: [36×1 double]

ProbA: [] ProbB: []

nSV: [2×1 double]

sv_coef: [36×1 double]

SVs: [36×2 double]

Finding the Decision Function

libsvm

- **w**: maybe infinite variables
- The dual problem

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le C, i = 1, \ldots, l \\ & \mathbf{y}^T \boldsymbol{\alpha} = 0, \end{aligned}$$

where $Q_{ij} = y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ and $\mathbf{e} = [1, \ldots, 1]^T$

Corresponds to (Bishop 7.32) With y=t

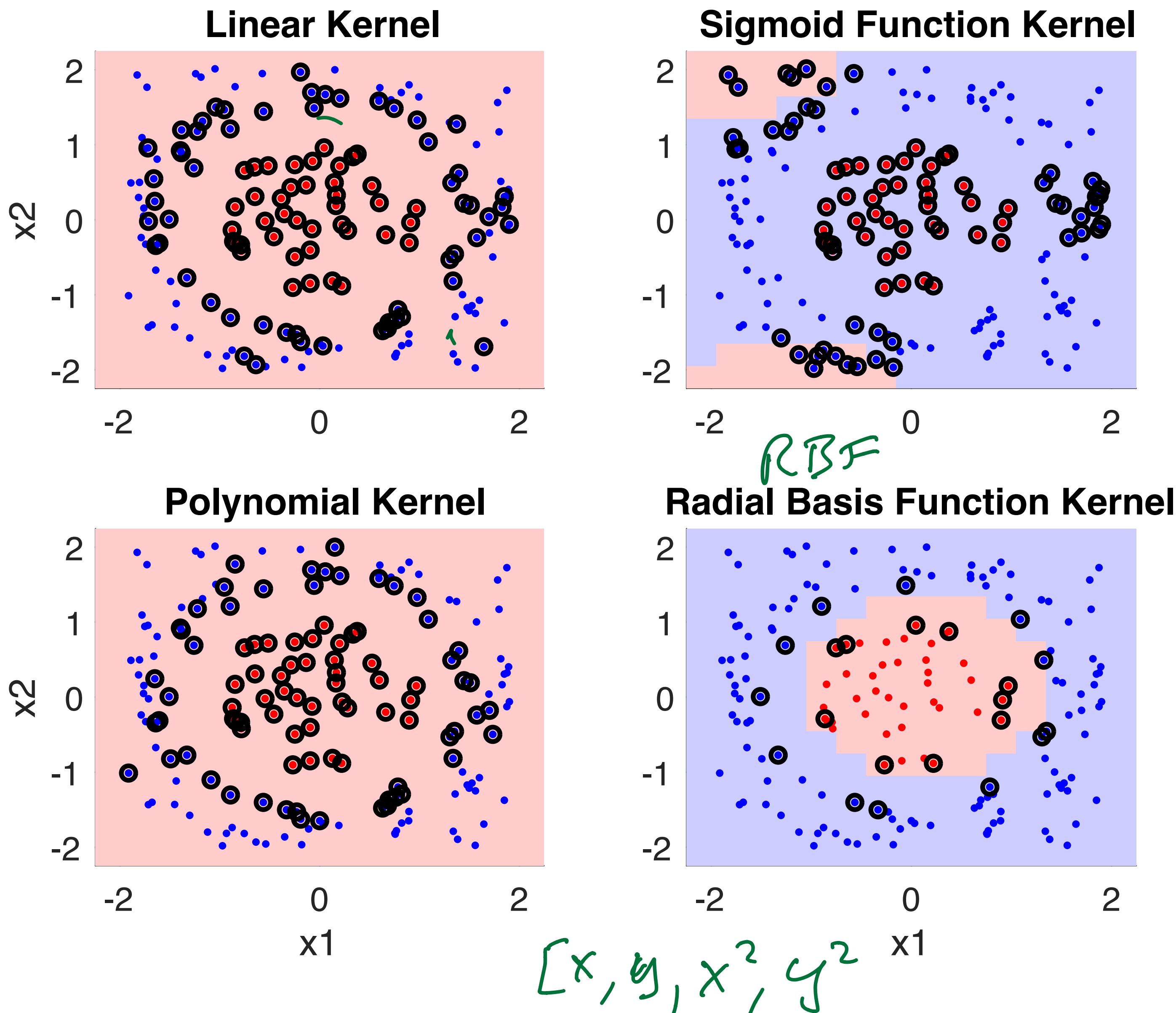
- At optimum

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \phi(\mathbf{x}_i)$$

- A finite problem: #variables = #training data

Using these results to eliminate $\mathbf{w}$, b, and $\{\xi_n\}$ from the Lagrangian, we obtain the dual Lagrangian in the form

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{7.32}$$


Linear Kernel
Sigmoid Function Kernel
Polynomial Kernel
Radial Basis Function Kernel
x1
x2
-2
-1
0
1
2
RBF
[x, y, x², y²

Gaussian Kernels

- Gaussian Kernel

$$k(x, x') = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{x}')\right)$$

Diagonal Σ: (this gives ARD)

$$k(x, x') = \exp\left(-\frac{1}{2}\sum_{i}^{N}\frac{(x_i - x_i')^2}{\sigma_i^2}\right)$$

Isotropic σ_i^2 gives an RBF

$$k(x, x') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\sigma^2}\right)$$

Can be inner product in infinite dimensional space

Assume $x \in R^1$ and $\gamma > 0$.

$$e^{-\gamma\|x_i-x_j\|^2} = e^{-\gamma(x_i-x_j)^2} = e^{-\gamma x_i^2+2\gamma x_i x_j-\gamma x_j^2}$$

$$=e^{-\gamma x_i^2-\gamma x_j^2}\left(1+\frac{2\gamma x_i x_j}{1!}+\frac{(2\gamma x_i x_j)^2}{2!}+\frac{(2\gamma x_i x_j)^3}{3!}+\cdots\right)$$

$$=e^{-\gamma x_i^2-\gamma x_j^2}\left(1\cdot 1+\sqrt{\frac{2\gamma}{1!}}x_i\cdot\sqrt{\frac{2\gamma}{1!}}x_j+\sqrt{\frac{(2\gamma)^2}{2!}}x_i^2\cdot\sqrt{\frac{(2\gamma)^2}{2!}}x_j^2\right.$$

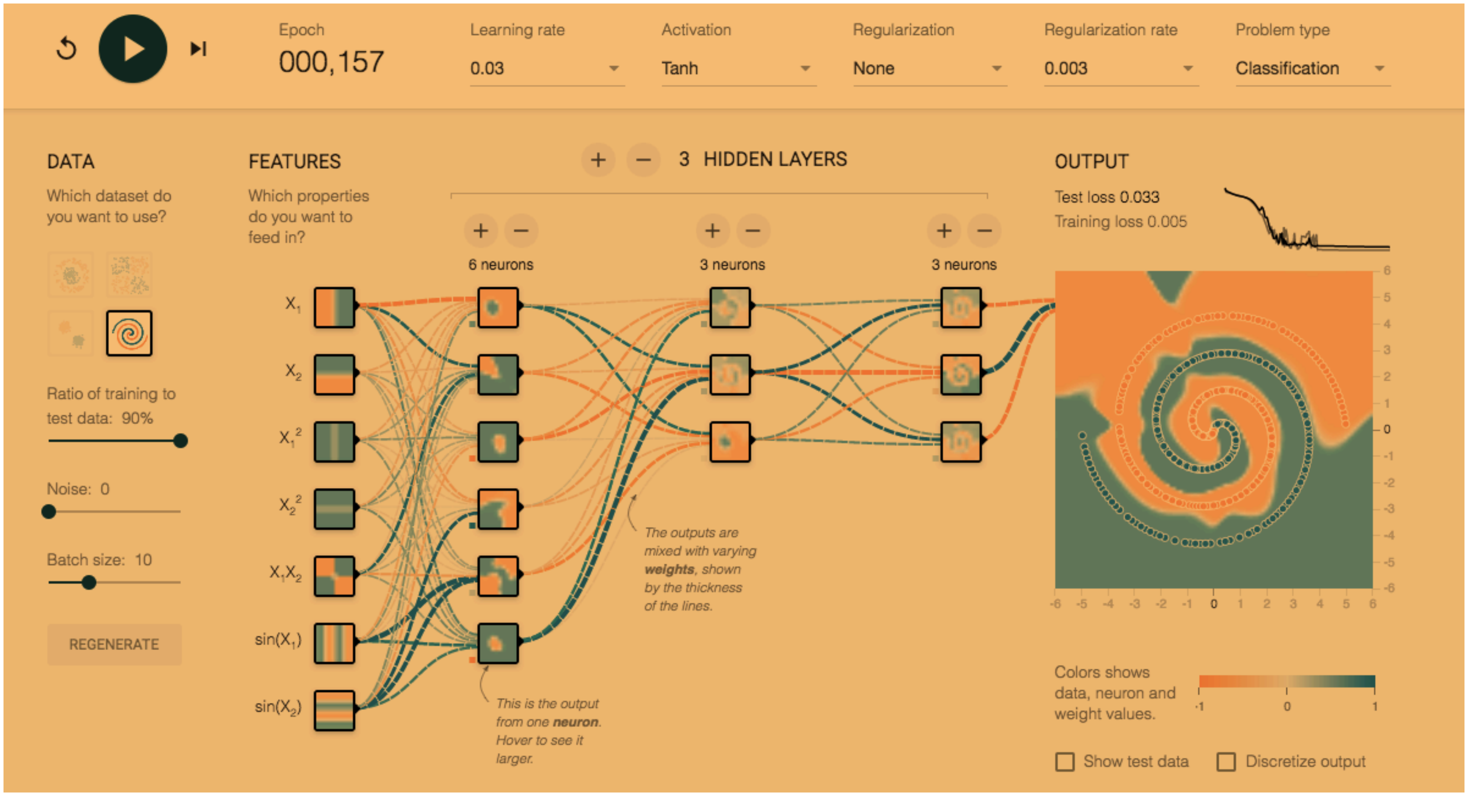
$$\left.+\sqrt{\frac{(2\gamma)^3}{3!}}x_i^3\cdot\sqrt{\frac{(2\gamma)^3}{3!}}x_j^3+\cdots\right)=\phi(x_i)^T\phi(x_j),$$

where

$$\phi(x)=e^{-\gamma x^2}\left[1,\sqrt{\frac{2\gamma}{1!}}x,\sqrt{\frac{(2\gamma)^2}{2!}}x^2,\sqrt{\frac{(2\gamma)^3}{3!}}x^3,\cdots\right]^T.$$

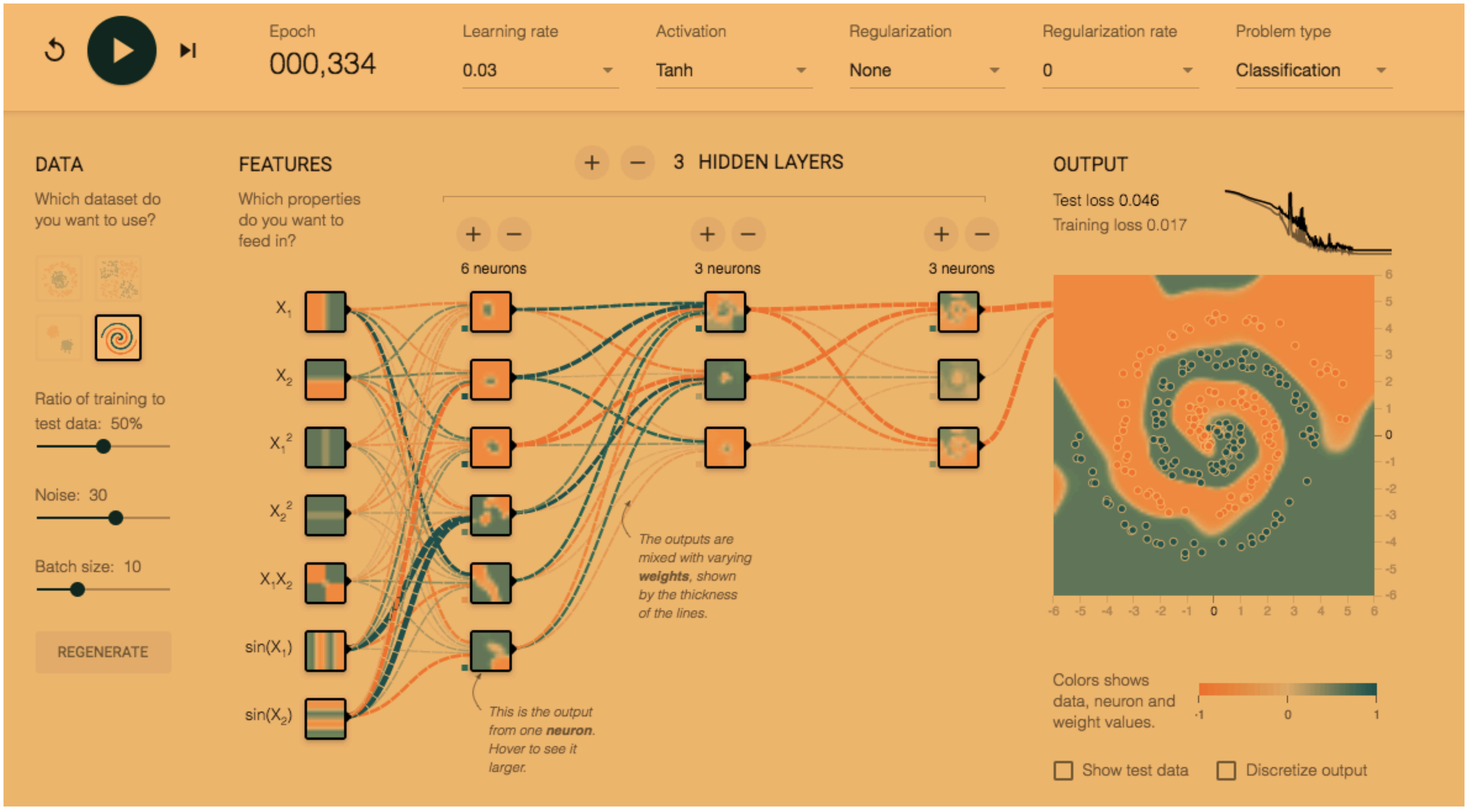
Tensorflow Playground

1. Fitting the spiral with default settings fail due to the small training set. The NN will fit to the training data which is not representative of the true pattern and the network will **generalize** poorly. Increasing the ratio of training to test data to 90% the NN finds the correct shape (1st image).



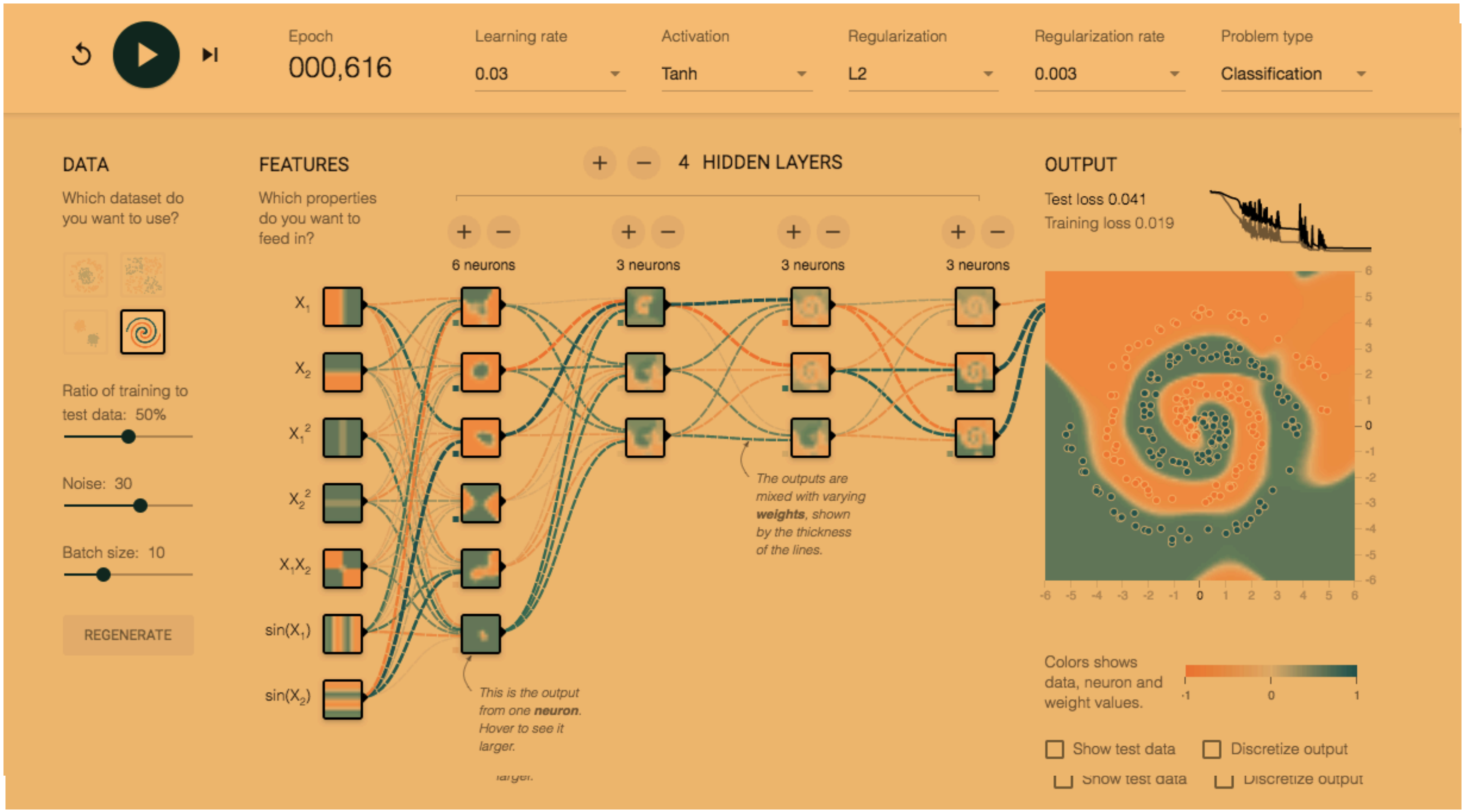
Tensorflow Playground

You can fix the generalization problem by adding **noise** to the data. This allows the small training set to generalize better as it reduce **overfitting** of the training data (2nd image).



Tensorflow Playground

Adding an additional hidden layer the NN fails to classify the shape properly. **Overfitting** once again becomes a problem even after you've added noise. This can be fixed by adding appropriate **L2 regularization** (third image).



- NOT USED